Dynamic variable ordering

- Static variable ordering algorithms assume that the variable at depth *i* is always x_i
- Assumption breaks down with dynamic variable ordering

$\Rightarrow$ Maintain a clear distinction between *nodes* in the search tree and *variables*

  - decide whether each step of an algorithm indexes a node or a variable

Data structures

- nd_i
  - search node at level i
  - $nd_i \bullet var$ is the *index* of the variable at level i
  - $Con(i, j)$ is a constraint check between variables assigned at nodes nd_i and nd_j
- cl
  - current level in the search tree
  - nd_{cl} is the node at the current level
- *unassigned*
  - indices of unassigned variables

Backtrack search

```
procedure bcssp(n)
    consistent = true
    i = initialize()
    loop
        if consistent then (i, consistent) = label(i)
        else (i, consistent) = unlabel(i)
        if  i > n  then return "solution found"
        else if  i = 0  then return "no solution"
    endloop
end bcssp
```

Chronological backtracking with DVO: *label*

function *btvar-label*(*i*)

nd_{cl} •*var* = *i* and *unassigned* = *unassigned* \ {*i*}

for each $v_k \in CD_i$ **do**

Set $x_i = v_k$ and *consistent* = *true*

for *j* **from** *1* **to** *cl–1* **do**

if ¬ *Con*(*cl*, *j*) **then**

Remove v_k from CD_i and set *consistent* = *false*

Unassign x_i and **break** inner loop

endif

if *consistent* **then** *cl* = *cl* + *1* and **return** (*NextVar*(), *true*)

endfor

return (*i*, *false*)

end *btvar-label*

Chronological backtracking with DVO: *unlabel*

function *btvar-unlabel*(*i*)

$cl = cl - 1$

$h = nd_{cl} \bullet var$

$unassigned = unassigned \cup \{i\}$

$CD_i = D_i$

Remove current value assigned to x_h from CD_h

Unassign x_h

if CD_h is empty **then**

    return (*h*, *false*)

else

    return (*h*, *true*)

end *btvar-unlabel*

BJ and CBJ with DVO

- *max-check*$_i$
  - *i* is a *variable* index
  - *max-check*$_i$ stores a *node* index
- *conf*$_i$
  - *i* is a *variable* index
  - *conf*$_i$ is the set of past *nodes* with which x_i conflicts

Backmarking with DVO

- mcl_{ik}
  - i is a *variable* index and k is a *domain value* index
  - mcl_{ik} stores the index of the deepest *node* against which $x_i = v_k$ checked
- mbl_i
  - i is a *variable* index
  - mbl_i stores the index of the shallowest *node* that has changed since x_i was the current variable

Forward checking with DVO

When $x_i = v_k$ is attempted, the domain of each $x_j \in$ *unassigned* is filtered

- $reductions_j$
  - j is a *variable* index
- $past\text{-}fc_j$
  - j is a *variable* index
  - stack of past *nodes* that have checked against x_j
- $future\text{-}fc_i$
  - i is a *variable* index
  - stack of future *variables* against which x_i has checked

Minimum Remaining Values heuristic for DVO

- *NextVar*() selects the variable that has the minimum number of values consistent with all previously assigned variables
- Best if used in conjunction with FC
  - backward checking algorithm will do extra constraint checks to evaluate the MRV heuristic
- **Theorem:** MRV makes standard backjumping redundant

Experimental results

- Tables from Bacchus and van Run 1995

Handling non-binary constraints

- Conflict-directed backjumping
  - check each constraint in which current variable occurs and which is complete
  - $conf_i$ is updated with previous nodes that fail a constraint check
- Forward checking
  - check each constraint in which current variable occurs and which has one unassigned variable
  - domain of unassigned variable is filtered
  - $past\text{-}fc_j$ is a stack of sets of nodes